

Week 2.1

1. Callbacks,
2. Async functions
3. Promises
4. JS functions (map, filter)
5. Assignment

If you already know this, feel free to skip this video!

More of a revision video

No bounties today

Week 2.1

Before we start

We have covered this

Foundation

Foundation Javascript, async nature of JS

Node.js and its runtime

Databases (NoSQL/SQL)

Mongo and Postgres deep dive

TypeScript beginner to advance

Backend

Backend communication protocols

Express basic to advance

ORMs

Middlewares, routes, status codes, global catches

Zod

MonoRepos, turborepo

Serverless Backends

OpenAPI Spec

Autogenerated clients

Authentication using external libraries

Scaling Node.js, performance benchmarks

Deploying npm packages

Frontend

Reconcilers and Frontend frameworks

React beginner to advance

Internals of state, Context API

State management using recoil

CSS you need to know of, Flexbox, basic styling

Frontend UI frameworks, Deep dive into Tailwind

Containerization, Docker

Next.js

Custom hooks

In house auth using next auth

Basic Devops

Docker end to end

Deploying to AWS servers

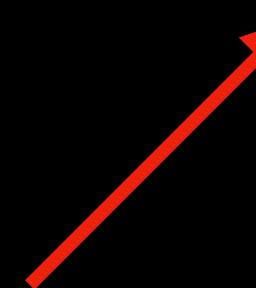
Newer clouds like fly/Remix

Nginx and reverse proxies

Projects

GSoC Project setting up and issue solving

Building Paytm/Wallet End to End



Week 2.1

Before we start

We have covered this

Foundation	Frontend
Foundation Javascript, async nature of JS Node.js and its runtime	Reconcilers and Frontend frameworks React beginner to advance
Databases (NoSQL/SQL) Mongo and Postgres deep dive TypeScript beginner to advance	Internals of state, Context API State management using recoil CSS you need to know of, Flexbox, basic styling Frontend UI frameworks, Deep dive into Tailwind Containerization, Docker Next.js Custom hooks In house auth using next auth
Backend	Basic Devops
Backend communication protocols Express basic to advance ORMs Middlewares, routes, status codes, global catches Zod MonoRepos, turborepo Serverless Backends OpenAPI Spec Autogenerated clients Authentication using external libraries Scaling Node.js, performance benchmarks Deploying npm packages	Docker end to end Deploying to AWS servers Newer clouds like fly/Remix Nginx and reverse proxies
Projects	GSOC Project setting up and issue solving Building Paytm/Wallet End to End



- For people who are struggling here -**
1. Try going through the offline videos
 2. This (esp promises) gets better with practise
 3. As the weeks progress, if you are consistent the syntax will fall into place
 4. Consistency is everything, don't drop off
 5. You need one eureka moment at which you get async programming and then the ride is smooth until React

There are 3 big humps in Full stack

1. Async nature of JS
2. React
3. JS to TS



We are here



**This is the gist of the hump
If you understand this, you are sorted
If not, there is a knowledge gap**

1.5 | Async, Await and Promises

Aynsc await syntax

In fact, all three are very similar (becomes more manageable as you move to the right)

Callback syntax

```
index.js > f main > ...
1 v function kiratsAsyncFunction(callback) {
2 v   // do some async logic here
3 v   callback("hi there!")
4 v }
5
6 v async function main() {
7 v   kiratsAsyncFunction(function(value) {
8 v     console.log(value);
9 v   });
10 v }
11
12 main();
```

Promise (then) syntax

```
index.js > ...
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3 v     // do some async logic here
4 v     resolve("hi there!")
5 v   });
6 v   return p;
7 v }
8
9 v function main() {
10 v   kiratsAsyncFunction().then(function(value) {
11 v     console.log(value);
12 v   });
13 v }
14
15 main();
```

Async/await syntax

```
index.js > ...
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3 v     // do some async logic here
4 v     resolve("hi there!")
5 v   });
6 v   return p;
7 v }
8
9 v async function main() {
10 v   const value = await kiratsAsyncFunction();
11 v   console.log(value);
12 v }
13
14 main();
```

Pre-requisites - Functions, var, const, data types in JS

What did we cover until now?

Live session 1 - Computers, Basic JS syntax, **intro to callbacks**

Offline session 1 - Simple JS APIs (**Classes**, Date class, string functions, object functions)

Offline session 2 - Loops, functions and **callback functions**

Offline session 3 - Async functions, JS Architecture and **Promises**

What all we need to know Before we proceed to HTTP?

Definitely - callback

Good to know - JS Architecture, Promises

Callbacks

Look at the code on the right, can you understand it?

```
JS index.js > f sumOfSquares > ...

1
2  function square(n) {
3    return n * n;
4  }
5
6  function cube(n) {
7    return n * n * n;
8  }
9
10 function sumOfSquares(a, b) {
11  let square1 = square(a);
12  let square2 = square(b);
13  return square1 + square2;
14}
15
16 let ans = sumOfSquares(1, 2);
17 console.log(ans);
18
```

Callbacks

Look at the code on the right, can you understand it?

Now add a sumOfCube function to it

```
JS index.js > f sumOfSquares > ...

1
2  function square(n) {
3    return n * n;
4  }
5
6  function cube(n) {
7    return n * n * n;
8  }
9
10 function sumOfSquares(a, b) {
11  let square1 = square(a);
12  let square2 = square(b);
13  return square1 + square2;
14}
15
16 let ans = sumOfSquares(1, 2);
17 console.log(ans);
18
```

Callbacks

Look at the code on the right, can you understand it?

Now add a sumOfCube function to it

```
index.js > ...
2 v function square(n) {
3   return n * n;
4 }
5
6 v function cube(n) {
7   return n * n * n;
8 }
9
10 v function sumOfSquares(a, b) {
11   let square1 = square(a);
12   let square2 = square(b);
13   return square1 + square2;
14 }
15
16 v function sumOfCube(a, b) {
17   let square1 = cube(a);
18   let square2 = cube(b);
19   return square1 + square2;
20 }
21
22 let ans = sumOfCube(1, 2);
23 console.log(ans);
24
```

Callbacks

Problem? Code repetition

```
JS index.js > ...
+
2 v function square(n) {
3 |   return n * n;
4 }
5
6 v function cube(n) {
7 |   return n * n * n;
8 }
9
10 v function sumOfSquares(a, b) {
11 |   let square1 = square(a);
12 |   let square2 = square(b);
13 |   return square1 + square2;
14 }
15
16 v function sumOfCube(a, b) {
17 |   let square1 = cube(a);
18 |   let square2 = cube(b);
19 |   return square1 + square2;
20 }
21
22 let ans = sumOfCube(1, 2);
23 console.log(ans);
24
```

Callbacks

Problem? Code repetition
Can you create a single function (`squareOfSomething`) that does
This logic on a **function it gets as an input**

```
JS index.js > ...
1
2  function square(n) {
3    return n * n;
4  }
5
6  function cube(n) {
7    return n * n * n;
8  }
9
10 function sumOfSquares(a, b) {
11   let square1 = square(a);
12   let square2 = square(b);
13   return square1 + square2;
14 }
15
16 function sumOfCube(a, b) {
17   let cube1 = cube(a);
18   let cube2 = cube(b);
19   return cube1 + cube2;
20 }
21
22 let ans = sumOfCube(1, 2);
23 console.log(ans);
24
```

Callbacks

Problem? Code repetition

Can you create a single function (`squareOfSomething`) that does
This logic on a **function it gets as an input**

```
index.js > ...
2  function square(n) {
3    return n * n;
4  }
5
6  function cube(n) {
7    return n * n * n;
8  }
9
10 function sumOfSomething(a, b, callbackFn) {
11   let square1 = callbackFn(a);
12   let square2 = callbackFn(b);
13   return square1 + square2;
14 }
15
16 let ans = sumOfSomething(1, 2, square);
17 console.log(ans);
18
```

Generate 

Callbacks

Both are the same

```
function sumOfSomething(a, b, callbackFn) {  
  let square1 = callbackFn(a);  
  let square2 = callbackFn(b);  
  return square1 + square2;  
  
}  
  
let ans = sumOfSomething(1, 2, square);  
console.log(ans);
```

```
9  
0 v function sumOfSomething(a, b) {  
1   let square1 = square(a);  
2   let square2 = square(b);  
3   return square1 + square2;  
4 }  
5  
6 let ans = sumOfSomething(1, 2);  
7 console.log(ans);  
8
```

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

2. Async functions

2. Async functions

What is async? - Asynchronous

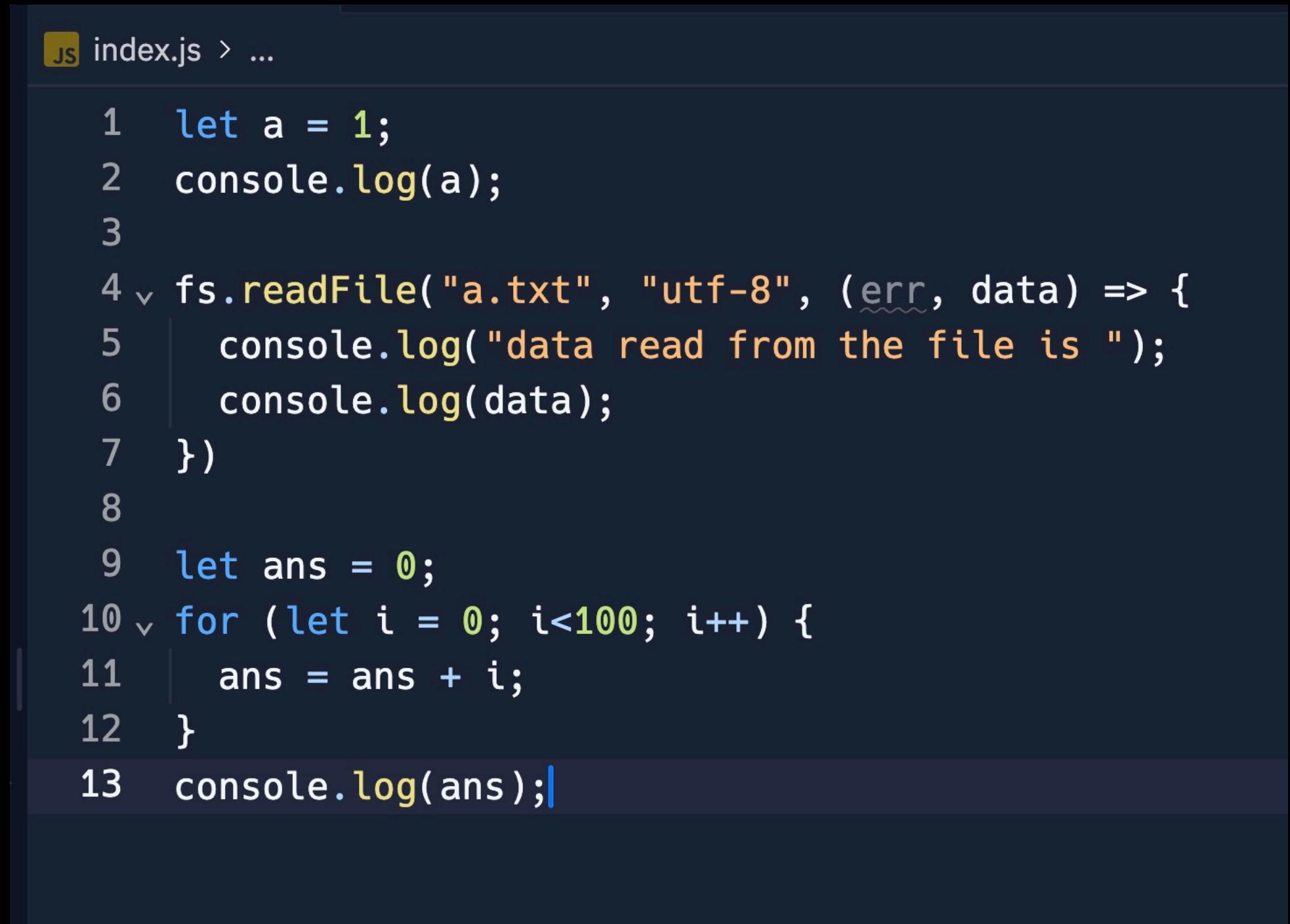
- 1. Your javascript thread doesn't have access to everything immediately**
- 2. There are some tasks it needs to **wait** for**

For example -

- 1. Reading a file**
- 2. Sending a network request**
- 3. A deliberate timeout**

2. Async functions

Lets see an **async** function call



The image shows a code editor window with a dark theme. The tab bar at the top has a yellow JS icon followed by "index.js > ...". The code itself is written in JavaScript and contains the following lines:

```
1 let a = 1;
2 console.log(a);
3
4 fs.readFile("a.txt", "utf-8", (err, data) => {
5   console.log("data read from the file is ");
6   console.log(data);
7 }
8
9 let ans = 0;
10 for (let i = 0; i<100; i++) {
11   ans = ans + i;
12 }
13 console.log(ans);
```

The line "13 console.log(ans);" is highlighted with a dark blue background.

2. Async functions

Lets see an async function call

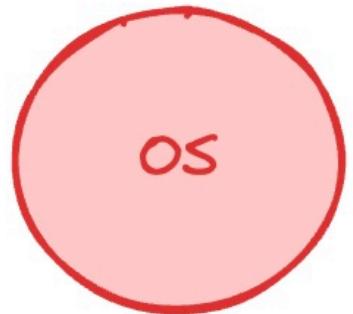


2. Async functions

Lets see an async function call

JS Thread run is idle

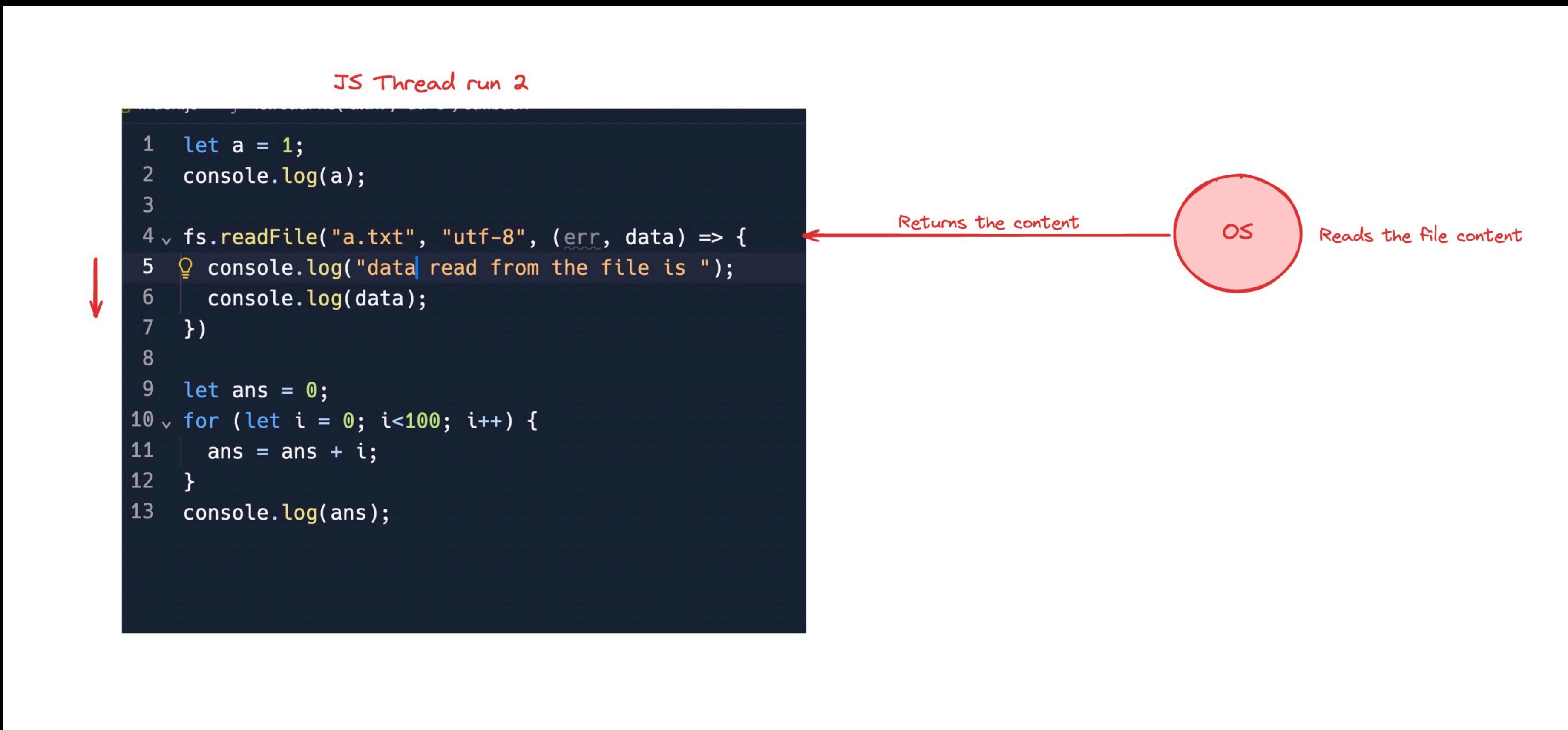
```
1 let a = 1;
2 console.log(a);
3
4 fs.readFile("a.txt", "utf-8", (err, data) => {
5   console.log("data read from the file is ");
6   console.log(data);
7 }
8
9 let ans = 0;
10 for (let i = 0; i<100; i++) {
11   ans = ans + i;
12 }
13 console.log(ans);
```



Reads the file content

2. Async functions

Lets see an async function call



2. Async functions

Lets see it on loupe

<http://latentflip.com/loupe/>

```
1
2  console.log("Hi!");
3
4 v setTimeout(function timeout() {
5   |   console.log("Click the button!");
6   }, 5000);
7
8
9  let ans = 0;
10 v for (let i = 0; i<10; i++) {
11   |   ans = ans + i;
12 }
13 console.log(ans);
```

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

Promises

Lets start with a question

Promises

What are promises?

1. Just syntactical sugar
2. Just a more readable way to write async functions

Can you write JS without them

Yes - Just use callbacks



Promises

Lets start with a question

How would you create your own async function?

Promises

Approach #1 (Wrapping another async fn)



The image shows a screenshot of a code editor with a dark theme. The file is named "index.js". The code contains two functions: "myOwnsetTimeout" which wraps another function with setTimeout, and "myOwnsetTimeout" which logs "hi there" to the console after a 1000ms delay.

```
1 v function myOwnsetTimeout(fn, duration) {
2   |   setTimeout(fn, duration);
3   }
4
5 v myOwnsetTimeout(function() {
6   |   console.log("hi there");
7   }, 1000)
```

Promises

Approach #1 (Wrapping another async fn)

```
JS index.js > ...
1 v function myOwnSetTimeout(fn, duration) {
2   |   setTimeout(fn, duration);
3   }
4
5 v myOwnSetTimeout(function() {
6   |   console.log("hi there");
7   }, 1000)
```

This approach uses a callback
You have created a function where other people can send a callback
This is good, but could lead to callback hell

Promises

Approach #1 (Wrapping another async fn)

```
1 function myOwnSetTimeout(fn, duration) {  
2   setTimeout(fn, duration);  
3 }  
4  
5 myOwnSetTimeout(function() {  
6   console.log("log the first thing");  
7   myOwnSetTimeout(function() {  
8     console.log("log the second thing");  
9   }, 2000)  
10 }, 1000)
```

This approach uses a callback
You have created a function where other people can send a callback
This is good, but could lead to **callback hell**

What if I tell you -
Create a function that logs something after 1s
And then waits for 2 seconds to log another thing

<https://gist.github.com/hkirat/70eecb2d8ed0b5ef6a393e4f1e5369e1>

Promises

Approach #1 (Wrapping another async fn)

```
1 v function myOwnSetTimeout(fn, duration) {  
2   setTimeout(fn, duration);  
3 }  
4  
5 v myOwnSetTimeout(function() {  
6   console.log("log the first thing");  
7 v   myOwnSetTimeout(function() {  
8     console.log("log the second thing");  
9   }, 2000)  
10 }, 1000)
```

This approach uses a callback

You have created a function where other people can send a callback

This is good, but could lead to **callback hell**

What if I tell you -

Create a function that logs something after 1s
And then waits for 2 seconds to log another thing

<https://gist.github.com/hkirat/70eecb2d8ed0b5ef6a393e4f1e5369e1>

Callbacks lead to unnecessary indentation

This is called **callback hell**

Promises

Approach #1 (Wrapping another async fn)

```
1 function myOwnSetTimeout(fn, duration) {  
2   setTimeout(fn, duration);  
3 }  
4  
5 myOwnSetTimeout(function() {  
6   console.log("log the first thing");  
7   myOwnSetTimeout(function() {  
8     console.log("log the second thing");  
9   }, 2000)  
10 }, 1000)
```

This approach uses a callback

You have created a function where other people can send a callback

This is good, but could lead to **callback hell**

What if I tell you -

Create a function that logs something after 1s
And then waits for 2 seconds to log another thing

<https://gist.github.com/hkirat/70eecb2d8ed0b5ef6a393e4f1e5369e1>

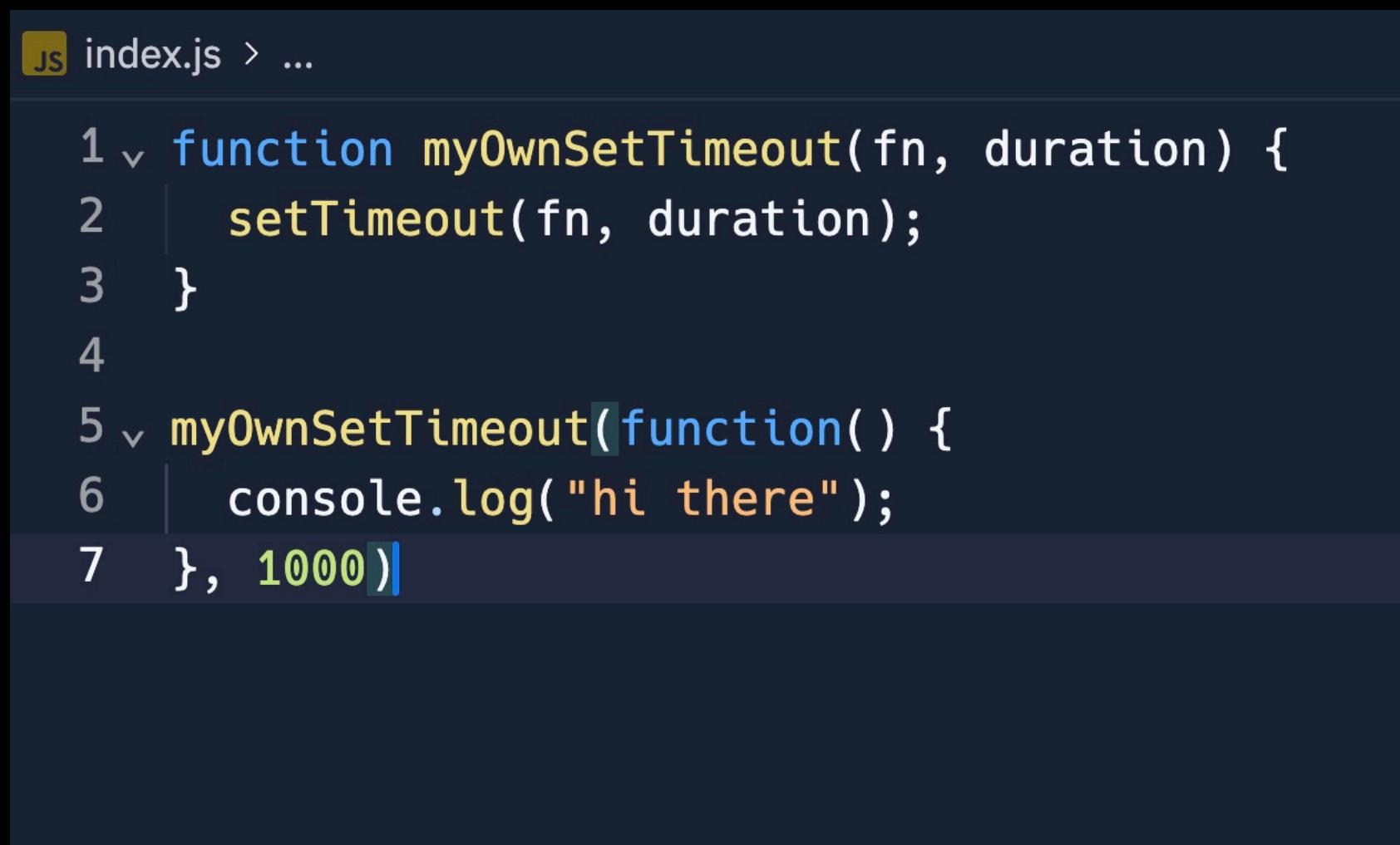
Callbacks lead to unnecessary indentation

This is called **callback hell**

Lets see how promises fix this

Promises

Approach #1 (Wrapping another async fn)



The image shows a screenshot of a code editor with a dark theme. The tab bar at the top has a yellow JS icon followed by the text "index.js > ...". The main editor area contains the following code:

```
1 v function myOwnSetTimeout(fn, duration) {  
2   |   setTimeout(fn, duration);  
3 }  
4  
5 v myOwnSetTimeout(function() {  
6   |   console.log("hi there");  
7 }, 1000)|
```

The code defines a function `myOwnSetTimeout` that wraps the built-in `setTimeout` function. It takes two arguments: a function `fn` and a duration. Inside the function, it calls `setTimeout(fn, duration)`. Below this, a call to `myOwnSetTimeout` is shown with an anonymous function that logs "hi there" to the console and a duration of 1000.

Promises

Approach #1
(Wrapping another async fn)

```
JS index.js > ...
1 v function myOwnSetTimeout(fn, duration) {
2 |   setTimeout(fn, duration);
3 }
4
5 v myOwnSetTimeout(function() {
6 |   console.log("hi there");
7 }, 1000)|
```

Approach #2
(Using promises)

```
JS index.js > f myOwnSetTimeout > ...
1 v function myOwnSetTimeout(duration) {
2 v   let p = new Promise(function (resolve) {
3 |     // after 1 second, call resolve
4 |     setTimeout(resolve, 1000);
5 |   });
6 |   return p;
7 }
8
9 myOwnSetTimeout(1000)
10 v   .then(function () {
11 |   console.log("log the first thing");
12 | });
13
```

<https://gist.github.com/hkirat/9a7a0ef9ad6788f645497a2cd2b92106>

Promises

Approach #1 (Wrapping another async fn)

```
index.js > ...
1 function myOwnsetTimeout(fn, duration) {
2   setTimeout(fn, duration);
3 }
4
5 myOwnsetTimeout(function() {
6   console.log("hi there");
7 }, 1000)
```

Approach #2 (Using promises)

```
index.js > f myOwnsetTimeout > ...
1 function myOwnsetTimeout(duration) {
2   let p = new Promise(function (resolve) {
3     // after 1 second, call resolve
4     setTimeout(resolve, 1000);
5   });
6   return p;
7 }
8
9 myOwnsetTimeout(1000)
10 .then(function () {
11   console.log("log the first thing");
12 });
13
```

Approach #3 (Async await)

```
index.js > ...
1 function myOwnsetTimeout(duration) {
2   let p = new Promise(function (resolve) {
3     // after 1 second, call resolve
4     setTimeout(resolve, 1000);
5   });
6   return p;
7 }
8
9 async function main() {
10   await myOwnsetTimeout(1000)
11   console.log("after");
12 }
13
14 main();
```

<https://gist.github.com/hkirat/9a7a0ef9ad6788f645497a2cd2b92106>

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

map

**Question - Convert an array of numbers into a new array with every element
Multiplied by 2**

map

**Question - Convert an array of numbers into a new array with every element
Multiplied by 2**

Do it yourself

Filter

Question - Convert an array of numbers into a new array with only the even elements

Do it yourself

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

Assignments